AD-A037 126 BROWN UNIV PROVIDENCE R I DIV OF APPLIED MATHEMATICS F/G 5/7
LINGUISTIC ABDUCTION MACHINES,(U)
JAN 76 U GRENANDER N00014-75-C-0461
UNCLASSIFIED NL

1 OF 1
AD A037 126

END
DATE FILMED
4-77

(15) Contract N00014-75-C-0461 New
Office of Naval Research and Brown University

(10)

(6) Linguistic Abduction Machines,

Working Paper No. 1

by

(10) Ulf Grenander
Division of Applied Mathematics
Brown University
Providence, Rhode Island

Date: Jan. 1976
(11)
(12) 8 p.

Let the syntactic variables be 1,2,3...v and words will be denoted x,y,z, etc. Introduce the matrix

$$P(x) = \{p_{ij}(x)\} \tag{1}$$

where $p_{ij}(x)$ is the probability of rewriting $i \rightarrow jx$, and the vector

$$r(x) = \{r_i(x)\} \tag{2}$$

where $r_i(x)$ is the probability of rewriting $i \rightarrow x$ (see Grenander's paper in Neyman Festschrift for further details and equation (3) used below).

When we search for the syntactic variables it may be best to organize the search from below following a suggestion by Henry Kucera. This means that we first try to group words into classes $c_1, c_2, \ldots, c_r$, then group classes into higher level classes and so on. It seems as if this would reduce the search effort drastically since the number of words n_w is much larger than the number of syntactic classes n_v.

When we do this we have to proceed by testing for linguistic equivalence similarly to method in paper on abduction machine. Two words x and y are said to be equivalent, written as $x \equiv y$, if uxv and uyv are either both grammatical or both non-grammatical, u and v arbitrary lexical strings.

The search will depend crucially on how difficult it is to separate x from y by equivalence when they are not equivalent. The trouble is that when we test with u and v, a negative answer

is enough to establish $x \not\equiv y$, but a positive answer is not. In principle we would have to go through all u and v.

Lemma 1. The statement $x \equiv y$ is the same as to say that P(uxv) and P(uyv) are both zero or not zero, all u and v strings.

Proof: For a given lexical string $S = x_1, x_2, \ldots x_n$ we get the probability

$$(3) \qquad P(S) = dP(x_1)P(x_2)\ldots P(x_{n-1})r(x_n)$$

where d is the vector (1,0,0,...0). We know from our earlier work however, that S being grammatical is the same as P(S) being positive, hence our statement correct, and we shall see that (3) can be used to clear up the situation more.

Introduce the function of two words x and y

$$(4) \qquad d(x,y) = \max_i \{\Sigma_j |p_{ij}(x)-p_{ij}(y)|\} + \max_i \{|r_i(x)-r_i(y)|\}.$$

Lemma 2. The function d is a pseudo distance

(a) $d \geq 0$, $d(x,x) = 0$

(b) $d(x,y) = d(y,x)$

(c) $d(x,z) \leq d(x,y)+d(y,z)$.

Proof: (a) and (b) are obvious. We have

$$(5) \quad d(x,z) = \max_i\{\Sigma_j|p_{ij}(x)-p_{ij}(z)|\} + \max_i\{|r_i(x)-r_i(z)|\}$$

$$\leq \max\{\Sigma_j|p_{ij}(x)-p_{ij}(y)|\} + \max\{\Sigma_j|p_{ij}(y)-p_{ij}(z)| +$$

$$\max_i|r_i(x)-r_i(y)| + \max_i|r_i(y)-r_i(z)|\} = d(x,y)+d(y,z).$$

Note however that $d(x,y) = 0$ does not imply $x=y$, it only means that $P(x) = P(y)$ and $r(x) = r(y)$. But using Lemma 1 this means that $x \equiv y$ so that the pseudo distance separates the words in the dictionary into equivalence classes. Also $x \equiv y$ does not imply $d(x,y) = 0$. It is also clear that $d \leq 2$ since $\sum_x \sum_j [p_{ij}(x)+r_i(x)] = 1$.

We can now get a bound on how difficult it is to separate x from y by the testing procedure mentioned above. We have using (3) for $S = x_1x_2,x_{r-1}xx_{r+1}\ldots x_n$ and $S' = x_1x_2\ldots x_{r-1}xx_{r+1}\ldots x_n$

$$(6) \quad P(S) - P(S') = d[P(x_1)\ldots P(x_{r-1})P(x)P(x_{r+1})\ldots r(x_n)$$

$$-dP(x_1)\ldots P(x_{r-1})P(y)P(x_{r+1})\ldots r(x_n)] =$$

$$= dA[P(x)-P(y)]Br(x_n).$$

The matrix $P(x)$ has now bounded by 1 since

$$(7) \quad \|P(x)\| \leq \max_i \sum_j p_{ij}(x) \leq 1$$

Hence $\|A\|$ and $\|B\| \leq 1$ so that

$$|P(S)-P(S')| \leq \|P(x)-P(y)\| \leq d(x,y) .$$

This was when v is not empty. If v is empty, so that the sentences end with x and y respectively, we get instead with a similar argument

$$(8) \quad |P(S)-P(S')| \leq \|r(x)-r(y)\| \leq d(x,y).$$

Hence we have

Theorem 1. The difference in test probabilities for two words x and y is bounded by

(9) $$|P(S)-P(S')| \leq d(x,y).$$

It is known at present how sharp this inequality is.

We now start the abduction from below and consider the dictionary $D = \{1,2,\ldots n_w\}$, to begin with consider as a single class called 1.

Partition Algorithm: After t sentences have been heard D has been partitioned into classes $c_1,c_2,\ldots,c_r$, mutually disjoint and exhaustive. When sentence No. t+1 is heard, S = uxy one word x appearing in it is picked (systematically or at random?) and replaced by another word y in the same class c(x). The following action is taken.

(a) if S' = uyv is grammatical nothing is done and the algorithm loops to the next sentence.

(b) if S' = uyv is not grammatical y is removed from class c(x) and we move to (c).

(c) start a new loop going through all the other classes c_i, $c_i \neq c(x)$. In each pick a word z (systematically or at random) and test for $x \equiv z$ as before. The first time the answer is positive move x to this class. Otherwise move to (d).

(d) create a new class c_{r+1} consisting of just x. Then move back to start.

A realization of this scheme may look like Figure 1. Note that the storage requirement for this scheme is very modest: a vector of length n_w whose entries are integers.

Of course step (c) can fail to establish a negative answer with positive probability. I guess, but this may be wrong, that the performance would improve by the following modification.

(c') same as (c) except that R words are picked (without replacement) from each class. If all lead to positive answers put x in this class, otherwise go on to the next class, etc. R could be called the replication number.

word \ t	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	2
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1
4	1	1	2	2	2	2	2	2
5	1	1	1	1	1	1	1	1
6	1	1	1	2	2	2	2	2
7	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	3	3
9	1	1	1	1	3	3	3	3
10	1	1	1	1	1	1	1	1
ρ =	1	1	2	2	3	3	3	3

Figure 1

Before implementing the partition algorithm for abduction, let us carry out an experiment as follows to gain some insight in its functionary and (lack of?) computational efficiently.

Define a probability measure p_x over the dictionary. Say that the true partition is

$$(10) \qquad \begin{cases} c_1 = \{1,2,\ldots n_1\} \\ c_2 = \{n_1+1, n_1+2, \ldots, n_1+n_2\} \\ \text{----} \end{cases}$$

and say that if $x \in c_i$, $y \in c_j$ then the test will give negative answer with a probability d_{ij}, depending only upon the class index. Let us choose

$$(11) \qquad \begin{cases} d_{ii} = 0 \quad \text{(this is not necessary, perhaps)} \\ 0 \leq d_{ij} \leq 1 \end{cases}$$

and let us also make d_{ij} into a distance. A simple choice would be

$$(12) \qquad d_{ij} = \frac{|i-j|}{\rho}$$

Simulate the procedure and count number of tests until true partition has been reached. The mean number of tests is an appropriate number of computational work required by the algorithm.

Finally a remark about a distance measure between two partitions described by the incidence matrices M and M'. Pick two elements x and y at random and look for the probability that $(x \underset{M}{\equiv} y) \neq (x \underset{M'}{\equiv} y)$.

Lemma 3. This probability is a distance.

Proof: We have

$$\delta(M,M') = \sum_{x,y} p_x p_y \{(x \underset{M}{\equiv} y) \neq (x \underset{M'}{\equiv} y)\} =$$

(13)

$$= \sum_{x,y} p_x p_y \, |m_{xy} - m'_{xy}|$$

This is a ℓ_1-metric so that the statement in the lemma is true.

It may be more natural to replace (4) by

$$(14) \qquad d(x,y) = \max_i \{ \sum_{j=1}^{n_{v+1}} |p_{ij}(x) - p_{ij}(y)| \} \ ,$$

where we have defined $p_{in_{v+1}}(x) = r_i(x)$.